# Resilin: Elastic MapReduce over Multiple Clouds

Anca Iordache, Christine Morin, Nikos Parlavantzas, Pierre Riteau


# Resilin: Elastic MapReduce over Multiple Clouds

**Anca Iordache, Christine Morin, Nikos Parlavantzas, Pierre Riteau**

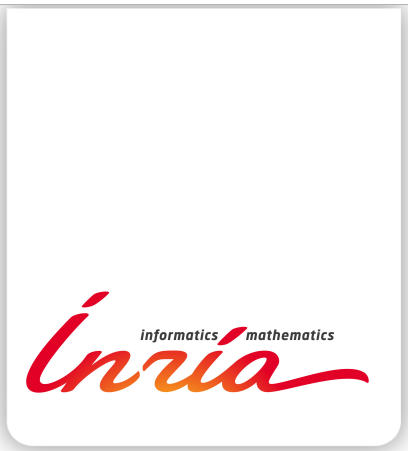
**RESEARCH REPORT**

**N° 8081**

September 2012

Project-Teams Myriads

ISSN 0249-6399 ISRN INRIA/RR--8081--FR+ENG




# Resilin: Elastic MapReduce over Multiple Clouds

Anca Iordache *, Christine Morin*, Nikos Parlavantzas†, Pierre Riteau‡

Project-Teams Myriads

Research Report n° 8081 — September 2012 — 17 pages

**Abstract:** The MapReduce programming model, introduced by Google, offers a simple and efficient way of performing distributed computation over large data sets. Although Google's implementation is proprietary, MapReduce can be leveraged by anyone using the free and open-source Apache Hadoop framework. To simplify the usage of Hadoop in the cloud, Amazon Web Services offers Elastic MapReduce, a web service enabling users to run MapReduce jobs. Elastic MapReduce takes care of resource provisioning, Hadoop configuration and performance tuning, data staging, fault tolerance, etc. This service drastically reduces the entry barrier to perform MapReduce computations in the cloud, allowing users to concentrate on the problem to solve. However, Elastic MapReduce is restricted to Amazon EC2 resources, and is provided at an additional cost. In this paper, we present Resilin, a system implementing the Elastic MapReduce API with resources from clouds other than Amazon EC2, such as private and scientific clouds. Furthermore, we explore a feature going beyond the current Amazon Elastic MapReduce offering: performing MapReduce computations over multiple distributed clouds. The evaluation of Resilin shows the benefits of running computations on more than one cloud. While not being the most efficient way to perform Hadoop computations, it solves the problem of resource availability and adds more flexibility regarding the type/price of resource.

**Key-words:** MapReduce, Hadoop, Cloud Computing, Multi-cloud environment

* INRIA Centre Rennes - Bretagne Atlantique, Rennes, France
† INSA, Rennes, France
‡ University of Chicago, USA

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

# Resilin : MapReduce élastique sur plusieurs nuages informatiques

**Résumé :**

Le modèle de programmation MapReduce, introduit par Google, offre un moyen simple et efficace de réaliser des calculs distribués sur de grandes quantités de données. Bien que la mise en oeuvre de Google soit propriétaire, MapReduce peut être utilisé librement avec l'environnement Hadoop. Pour simplifier l'utilisation de Hadoop dans les nuages informatiques, Amazon Web Services offre Elastic MapReduce, un service web qui permet aux utilisateurs d'exécuter des applications MapReduce. Il prend en charge l'allocation de ressources, la configuration et l'optimisation de Hadoop, la copie des données, la tolérance aux fautes, etc. Ce service facilite l'exécution d'applications MapReduce dans les nuages informatiques, permettant ainsi aux utilisateurs de se concentrer sur la résolution de leur problème plutôt que sur la gestion de la plate-forme d'exécution. Elastic MapReduce est limité á l'utilisation de ressources fournies par Amazon EC2 et est proposé à un coût additionnel. Dans cet article, nous présentons Resilin, un système mettant en oeuvre l'API Elastic MapReduce avec des ressources provenant d'autres nuages informatiques que Amazon EC2, tels que les nuages privés ou communautaires. De plus, nous explorons une fonctionnalité nouvelle par rapport au service offert par Amazon Elastic MapReduce: l'exécution d'applications MapReduce sur plusieurs nuages géographiquement distribués. L'évaluation de Resilin montre les avantages liés à l'utilisation de plus d'un nuage pour l'exécution d'applications MapReduce. Bien qu'il ne fournisse pas la solution la plus efficace pour l'exécution d'applications MapReduce, Resilin résout le problème de la disponibilité des ressources et ajoute une plus grande flexibilité en ce qui concerne le type et le prix des ressources.

**Mots-clés :** MapReduce, Hadoop, Cloud Computing, Environnement multi-cloud

# 1 Introduction

The MapReduce programming model [11], proposed by Google in 2004, offers a simple way of performing distributed computation over large data sets. Users provide a map and a reduce function. The map function takes a set of input key/value pairs, and produces intermediate key/value pairs. The reduce function merges intermediate key/value pairs together to produce the result of the computation. This programming model became popular because it is simple yet expressive enough to perform a large variety of computing tasks. It can be applied to many fields, from data mining to scientific computing. This programming model is backed by a proprietary framework developed by Google that takes care of scheduling tasks to workers, sharing data through a distributed file system, handling faults, etc.

The Apache Hadoop [3] project develops a free and open-source implementation of the MapReduce framework. The Hadoop MapReduce framework works with the Hadoop Distributed File System (HDFS) [32], designed to be highly-reliable and to provide high throughput for large data sets used by applications. Hadoop is heavily used by companies such as Yahoo!, Facebook and eBay to perform thousands of computations per day over petabytes of data [34, 33]. However, managing a Hadoop cluster requires expertise, especially when scaling to a large number of machines. Moreover, users who want to perform MapReduce computations in cloud computing environments need to instantiate and manage virtual resources, which further complicates the process.

To lower the entry barrier for performing MapReduce computations in the cloud, Amazon Web Services provides Elastic MapReduce (EMR) [1]. Elastic MapReduce is a web service to which users submit MapReduce jobs. The service takes care of provisioning resources, configuring and tuning Hadoop, staging data, monitoring job execution, instantiating new virtual machines in case of failure, etc.

However, this service has a number of limitations. First, it is restricted to Amazon EC2 resources. Users are not able to use Elastic MapReduce with resources from other public clouds or from private clouds, which may be less expensive or even free of charge. This is especially true for scientists who have access to clouds administrated by their institution and dedicated to scientific computing [19, 30]. Moreover, Elastic MapReduce is provided for an hourly fee, in addition to the cost of EC2 resources. This fee ranges from 17% to 21% of the price of on-demand EC2 resources. It is impossible to use a different virtual machine image than the one provided by Amazon. Finally, some users may have to comply with data regulation rules, forbidding them from sharing data with external entities like Amazon.

In this paper, we present Resilin, a system implementing the Elastic MapReduce API with resources from other clouds than Amazon EC2, such as private and scientific clouds. Resilin allows users to perform MapReduce computations on different infrastructures than Amazon EC2, and offers more flexibility: users are free to select different types of virtual machines, different operating systems or newer Hadoop versions. The goal of Resilin is not only to be compatible with the Elastic MapReduce API. We also explore a feature going beyond the current Amazon Elastic MapReduce offering: performing MapReduce computations over multiple distributed clouds.

This paper is organized as follows. Section 2 presents the Amazon Elastic MapReduce service in more details. Section 3 covers the architecture and implementation of Resilin. Section 4 presents our experiments and analyzes their results. Section 5 reviews related work. Finally, Section 6 concludes and discusses future work.

## 2 Amazon Elastic MapReduce

Amazon Elastic MapReduce [1] is one of the products of Amazon Web Services. After signing up and providing a credit card number, users can submit MapReduce jobs through the AWS management console (a web interface), through a command line tool, or by directly calling the Elastic MapReduce API (libraries to access this API are available for several programming languages, such as Java and Python).

Users execute Hadoop jobs with Elastic MapReduce by submitting job flows, which are sequences of Hadoop computations. Before starting the execution of a job flow, Elastic MapReduce provisions a Hadoop cluster from Amazon EC2 instances. Each job flow is mapped to a dedicated Hadoop cluster: there is no sharing of resources between job flows. A Hadoop cluster created by Elastic MapReduce can be composed of three kinds of nodes:

- the master node, which is unique, acts as a meta data server for HDFS (by running the NameNode service) and schedules MapReduce tasks on other nodes (by running the JobTracker service),
- core nodes provide data storage to HDFS (by running the DataNode service) and execute map and reduce tasks (by running the TaskTracker service),
- task nodes execute map and reduce tasks but do not store any data.

By default, a Hadoop cluster created by Elastic MapReduce is composed of one master node and several core nodes[1]. At any time during the computation, a Hadoop cluster can be resized. New core nodes can be added to the cluster, but core nodes cannot be removed from a running job flow, since removing them could lead to HDFS data loss. Task nodes can be added at any time, and removed without risk of losing data: only the progress of MapReduce tasks running on the terminated task nodes will be lost, and the fault tolerance capabilities of Hadoop will trigger their re-execution on other alive nodes.

After the Hadoop cluster has booted, the service executes bootstrap actions, which are scripts specified by users. These actions allow users to customize an Elastic MapReduce cluster to some extent. Amazon provides several pre-defined bootstrap actions, to modify settings such as the JVM heap size and garbage collection behavior, the number of map and reduce tasks to execute in parallel on each node, etc. Users can also provide custom bootstrap actions by writing scripts and uploading them in S3. Once a Hadoop cluster is ready for computation, Elastic MapReduce starts executing the associated job flow. A job flow contains a sequence of steps, which are executed in order. Internally, each step corresponds to the execution of a Hadoop program (which can itself spawn multiple Hadoop jobs). After all steps are executed, the cluster is normally shut down. Users can ask for the cluster to be kept alive, which is useful for debugging job flows, or for adding new steps in a job flow without waiting for cluster provisioning and paying extra usage (in Amazon EC2, instance cost is rounded up to the hour, which means a 1-minute and a 59-minute job flow cost the same price).

Typically, input data is fetched from Amazon S3, a highly scalable and durable storage system provided by Amazon. Intermediate data is stored in HDFS, the Hadoop Distributed File System. Output data is also saved in Amazon S3, since the termination of a Hadoop cluster causes its HDFS file system to be destroyed and all the data it contains to become unavailable.

[1] If a user requests a Hadoop cluster composed of only one node, the same virtual machine performs the roles of master node and core node.

# 3 Architecture and Implementation

Like the Amazon Elastic MapReduce service, Resilin provides a web service acting as an abstraction layer between users and Infrastructure as a Service (IaaS) clouds. Figure 1 presents how Resilin interacts with other components of a cloud infrastructure. Users execute client programs supporting the Elastic MapReduce API to interact with Resilin (for instance, a Python script using the boto [4] library). Resilin receives Elastic MapReduce requests and translates them in two types of actions:

- interactions with an IaaS cloud using the EC2 API, to create and terminate virtual machine instances,
- remote connections to virtual machines using SSH, to configure Hadoop clusters and execute Hadoop programs.

To retrieve input data and store output data, Hadoop clusters interact with a cloud storage repository through the S3 API.

Resilin implements most actions supported by the Amazon Elastic MapReduce API. Users are able to submit job flows (RunJobFlow), query job flow statuses (DescribeJobFlows), add new steps to an existing job flow (AddJobFlowSteps), change the number of resources by adding or modifying instances groups (AddInstanceGroups/ModifyInstanceGroups), and terminate job flows (TerminateJobFlows).

Unlike Amazon, Resilin allows users to remove core nodes from the Hadoop cluster. This feature is going to be described in more details in section 3.2.

Currently, most applications types supported by Amazon Elastic MapReduce are available in Resilin. Users using Resilin are able to launch JAR-based and streaming Hadoop jobs. Data analysis is also supported by writing job flows using Apache Hive and Apache Pig, two SQL-like languages.

We now describe how each type of Elastic MapReduce action is handled. When a new job flow request (RunJobFlow) is received by Resilin, its parameters are validated, and the service contacts an EC2-compatible IaaS cloud on behalf of the user to provision a Hadoop cluster. By leveraging the EC2 API, Resilin can make use of resources from clouds managed by open-source toolkits supporting the EC2 API, such as Nimbus [24], OpenNebula [26], OpenStack [27] and Eucalyptus [25].

Once a Hadoop cluster is provisioned, Resilin connects to the virtual machines, using SSH, to execute bootstrap actions specified by the user, configure the Hadoop cluster (specifying the HDFS NameNode address, the MapReduce JobTracker address, etc.), and start the Hadoop daemons.

After Hadoop has been configured, the associated job flow is started and each step in the job flow is executed by invoking the `hadoop` command on the master node. Each step specifies the locations of its input and output data. These references can be S3 URLs: Hadoop supports reading and writing directly to Amazon S3. It would be technically possible to configure the Hadoop clusters created by Resilin to access input data from Amazon S3, like in Amazon Elastic MapReduce. However, this is ineffective for two reasons. First, bandwidth limitations between Amazon S3 and the private or scientific IaaS cloud running the Hadoop cluster drastically limit performance. Second, outgoing traffic from Amazon S3 is charged to customers, which would incur a high cost when performing MapReduce computations on big data sets.

We assume that clouds used by Resilin have a storage service available. Both Nimbus [24] and Eucalyptus [25] provide their own S3-compatible cloud storage (respectively called Cumulus [5] and Walrus). Furthermore, Cumulus can be installed as a standalone system, making it available

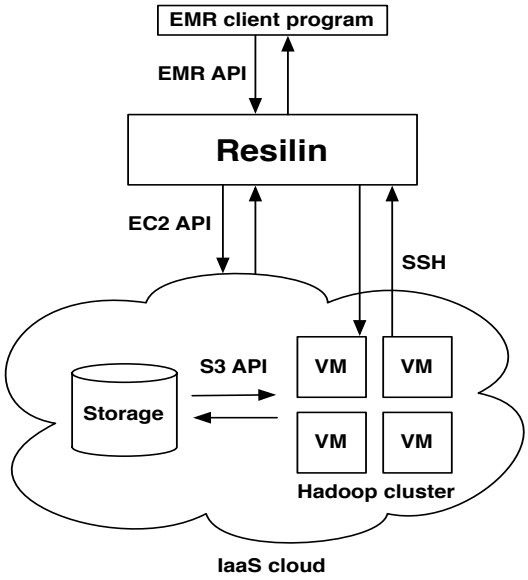



Figure 1: Overview of Resilin and its interactions with other cloud components

for any cloud deployment. Resilin can use this kind of repository to provide Hadoop with input data and to store output data. To make it possible, we extended the S3 protocol support of Hadoop 0.20.2. In addition to being able to specify URLs in the form of `s3n://bucket/key`, users can also provide URLs such as `cumulus://bucket.host:port/key`. When such URLs are detected, the library used by Hadoop to interact with S3 (JetS3t) is set up to contact the Cumulus server running on `host:port`.

Users can run different applications. The first type, a custom JAR, is simply the location of a JAR and its required arguments.

When executed, this JAR submits jobs to Hadoop. The JAR URL can be a S3 or Cumulus location. In this case, Resilin first downloads the JAR to the master node, using the `hadoop fs -copyToLocal` command, as Hadoop does not support directly executing a JAR from a remote location.

The second type, streaming jobs, is defined by a mapper program, a reducer program, and the locations of input and output data. Both programs can be stored in S3 or Cumulus, and can be written in any programming language (Java, Ruby, Python, etc.): they apply their computation on data coming from standard input and stream their result to standard output. To run a streaming step, the Hadoop command is invoked with the hadoop-streaming JAR included in the Hadoop MapReduce framework.

We determined that Amazon made modifications to Hadoop to be able to fetch the mapper and reducer programs from S3 and add them to the Hadoop distributed cache (in order to provide them to all nodes). We did not make such modifications to the Hadoop source code, and currently rely on bootstrap actions to download the mapper and reducer programs to the master node. Other types of job flows involve data analysis systems as Apache Hive and Pig. These systems need to be installed in the virtual cluster. For running installation scripts as jobflow steps, we use a tool provided by Amazon, script-runner.jar. It is requiring as argument the program to execute. Using this tool, scripts for installing and configuring new systems in

Hadoop can be executed. For example, when running a Hive or Pig script, a first step of the jobflow is the installation of the system and then the execution of the targeted program.

Resilin monitors the execution of the Hadoop computation. When the execution of a step is finished, the status of the job flow is updated, and the next step starts running. When all steps have been executed, the job flow is finished, which triggers termination of the cluster (unless the user requested the cluster to be kept alive).

Resilin is implemented in Python. It uses the Twisted [35] framework for receiving and replying to HTTP requests, the boto [4] library to interact with clouds using the EC2 API, and the paramiko [29] library for executing commands on virtual machines using SSH.

Also, it comes with its own EMR client that eases job launching and contains installers for Apache Hive, Apache Pig and for setting cluster parameters. These scripts are equivalent to the ones used in Amazon EMR with some differences. Amazon EMR is using some libraries to improve Hive and Pig while the scripts offered by Resilin installs them as they are provided by Apache, with no extra feature.

Yet, Resilin's client is not mandatory, any EMR client tool can be used instead.

## 3.1 Multi-Cloud Job Flows

Besides targeting compatibility with the Amazon Elastic MapReduce API, we are experimenting with an additional feature in Resilin: the ability to execute job flows with resources originating from multiple clouds. With Amazon Elastic MapReduce, executing MapReduce computations across multiple Amazon EC2 regions does not present a lot of interest. The large number of resources available in each region makes it possible to deploy large Hadoop clusters in a single data center[2]. However, in the usage context of Resilin, many users would have access to several moderately sized private or scientific clouds. For example, the FutureGrid [14] project, a distributed, high-performance testbed in the USA, contains 4 Nimbus clouds, each having from 120 to 328 processor cores. Federating computing power from several such clouds to create large scale infrastructures has been proposed as the *sky computing* approach [18].

Although creating MapReduce clusters distributed over several clouds may not be efficient because of the large amounts of wide-area data transfer that it generates [7], it is interesting for some types of MapReduce jobs. For example, Matsunaga et al. [22] have shown that multi-cloud BLAST computations with MapReduce can scale almost linearly.

Resilin supports multi-cloud job flows by allowing users to dynamically add new resources from different clouds to a running job flow. As an example, let us assume that a user has access to two different clouds, *Cloud A* and *Cloud B*, and that *Cloud A* is her default choice. After a job flow has been started on *Cloud A*, Resilin will accept requests for adding new instances (AddInstanceGroup) with an instance type also specifying the cloud to use: instead of `m1.small`, the instance type of the request would be `m1.small@CloudB`. After new virtual machines are provisioned from *Cloud B*, they are configured to become resources of the cluster running in *Cloud A*. This addition is managed seamlessly by Hadoop. When Hadoop daemons are started in the newly provisioned virtual machines of *Cloud B*, they contact the master node in *Cloud A* to join the cluster and start receiving MapReduce tasks to execute.

[2]When scaling to very large clusters, Amazon EC2 can start running out of resources. In March 2011, the Cycle Computing company published details about how they provisioned a 4096-core cluster in the Amazon EC2 US East region. When they scaled up the number of resources, 3 out of the 4 availability zones of the region reported Out of Capacity errors.

### 3.2 Cluster Elasticity

During execution time, the cluster can be resized by making requests (ModifyInstanceGroups) to Resilin with the new number of instances in the group (core group or task group). Therefore, new instances can be added to the cluster or existing ones removed. Scaling up behaves the same as the AddInstanceGroups request, Resilin starts and configures new virtual machines that will join the cluster. Scaling down, though, has a different execution flow depending on the group to be resized. When resizing a task group, Resilin is simply stopping the mapreduce daemon and shutting down the virtual machine. But, in the case of core group, there is extra work to be done due to the data stored by these virtual machines. Therefore, before shutting down the virtual machines, the data must be migrated to the remaining machines. To accomplish this, Resilin asks Hadoop to decomission those machines. This process is successful only if there is enough space on the remaining machines to store the data from the removed virtual machines.

## 4 Evaluation

We validated Resilin features on a single cloud comparing it with Amazon EMR for the execution of all kinds of applications supported. We also evaluated the performance obtained by Map/Reduce applications using resources from multiple clouds for both static and elastic virtual clusters. We performed our experiments using the Grid'5000 experimentation platform [6]. In our experiments we used Nimbus Infrastructure version 2.8 [24] and OpenNebula Infrastructure version 3.4.

We used a number of applications from the *HiBench* [16] benchmark (*wordcount*, *terasort* and *PageRank*).Word Count reports the number of occurrences of each word in a text collection. It is representative for real-world applications, transforming a large input into an intermediate representation from which interesting data is filtered. The mappers only emit key/value pairs with the *<word, 1>* format whose values are going to be summed up by the reducers. This application is executed over a 100 GB dataset generated using *RandomWriter* sample from Hadoop distribution.

TeraSort is the most well-known benchmark for Hadoop. Its goal is to sort an amount of data as fast as possible. To run this application, 10 GB of data was generated using the teragen sample included in Hadoop distribution. PageRank is a representative use of MapReduce. It is an algorithm for link analysis, used in search engines for computing the rank of web pages according to the number of reference links. The workload consists of a number of job flows with 64 map tasks and 32 reduce tasks processing 1050000 automatically generated Web pages.

In our measurements, we distinguish the virtual cluster deployment time from the application execution time. The deployment phase consists in provisioning a virtual cluster from the IaaS cloud manager, starting the virtual cluster and configuring all the virtual cluster's VMs. The execution of a Map/Reduce application is split in three phases: (1) transfer of input data from the cloud storage (S3 for Amazon EMR, Cumulus for Resilin) to HDFS, (2) execution of the Map/Reduce application, (3) transfer of the output data from HDFS to the cloud storage. In Amazon, S3 is a distributed storage system while in Resilin, we use Cumulus, which provides the S3 interface on top of a centralized NFS server. In all our experiments, the application execution time reported corresponds to the execution time of the second phase only. Indeed, our benchmarks generate the input data directly in HDFS. We consider an application has finished its execution when the output data is available in HDFS.

| Instance | Amazon EC2 | Nimbus |
|---|---|---|
| Virtual cores | 2 | 2 |
| Memory | 1.7 GB | 2 GB |
| Disk space | 350 GB | 3.9GB |
| Physical CPU | Intel Xeon E5410 (2.33 GHz)<br>Intel Xeon E5506(2.13 GHz) | Intel Xeon E5420 QC<br>(2.5 GHz) |
| OS | 32-bit Debian Lenny 5.0.8 | 64-bit Debian Lenny 5.0.4 |

Table 1: Hardware and Software configuration of virtual machines

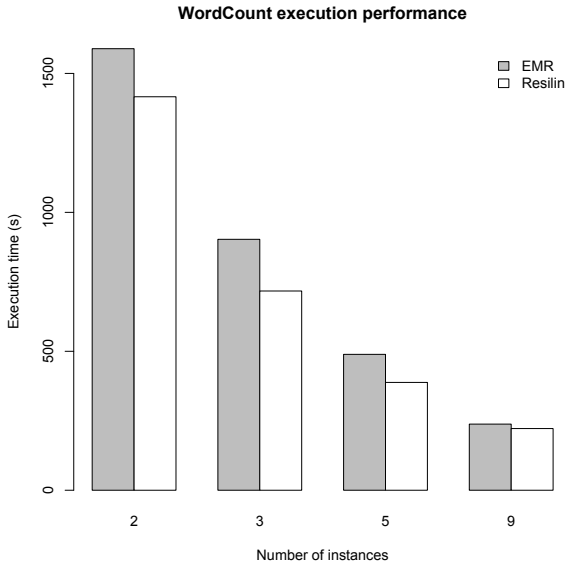



Figure 2: Execution time of WordCount on Amazon EMR and Resilin with different numbers of instances.

## 4.1 Comparison with Amazon EMR on a Single Cloud

We experimentally verified that Resilin correctly deploys and executes all applications types supported by Amazon EMR. We submitted the same job flow, composed of only one step, on both Amazon EMR and Resilin and compared the application execution times obtained with the two services (from the start of the requested Hadoop computation until its completion). Since Resilin is able to use different IaaS cloud managers and images for starting virtual machines, the deployment time is determined by the IaaS manager. In this experiment we ignored the deployment time and focused on application execution (independent of the IaaS providers and virtual machine image types).

We present the results obtained for the wordcount application taking as input 1.8GB of text files. In this experiment, Resilin exploits resources from a Nimbus cloud.

Table 1 reports the configuration of the virtual machines used by Amazon Elastic MapReduce and Resilin. They exhibit similar performance. However, we could not run experiments with VMs presenting exactly the same performance as VMs run on two different infrastructures.

Figure 2 presents the results of the execution time of Word Count with Amazon EMR and

| Instance | OpenNebula VM | Nimbus VM |
|---|---|---|
| Virtual cores | 1 (2.5 GHz) | 1 (2.5 GHz) |
| Memory | 2 GB | 2 GB |
| Disk space | 45 GB | 45GB |
| OS | Debian Squeeze 6.0.1<br>Linux 2.6.32 kernel | Debian Squeeze 6.0.1<br>Linux 2.6.32 kernel |

Table 2: Configuration of virtual machines from the clouds used in experiments

Resilin. For small cluster sizes, Resilin is faster than Amazon Elastic MapReduce. As the cluster size is increased, the difference of performance between Elastic MapReduce and Resilin becomes smaller. This is not surprising, since our Cumulus storage data transfer performance decreases when virtual machines client number increases while the S3 transfer remains the same with one or multiple clients.

## 4.2 Multi-cloud scenarios

For validating Resilin for Hadoop clusters comprising of resources from multiple clouds over a WAN, we compare the execution of the same job flows using Resilin on resources residing in one cloud and with resources provisioned from multiple clouds.

We used physical machines part of *genepi* cluster in Grenoble and *paradent* cluster in Rennes, for the deployment of one OpenNebula cloud and one Nimbus cloud. The machines from *genepi* cluster have 2 Intel Xeon E5420 QC processors (each with 4 cores running at 2.5 GHz), 8 GB of memory, and a Gigabit Ethernet connectivity. They were running a 64-bit Debian Squeeze 6.0 operating system with a Linux 2.6.32 kernel, and were using QEMU/KVM version 0.12.5 as hypervisor. On these machines, a Nimbus cloud was deployed.

For the OpenNebula cloud, machines from *paradent* cluster were used. These machines have similar performance with the *genepi* machines, 2 Intel Xeon L5420 QC processors (each with 4 cores running at 2.5 GHz), 16 GB of memory, and a Gigabit Ethernet connectivity. Their operating system and hypervisor are the same as *genepi* machines.

Table 2 reports the configuration of the virtual machines used with Resilin to perform experiments.

First, we evaluated the configuration time of the Hadoop cluster. Second, we compare the execution time, from submission of a Hadoop computation until its completion.

These experiments are run with various cluster sizes. In each experiment, we had a dedicated master node and we used the following numbers of core nodes: 6, 18, 30, 42.

In the multi cloud scenario, the bottleneck is the network I/O, the network data transfer rate between the two clouds being much smaller than the transfer rate between virtual machines in the same cloud.

## 4.3 Cluster Configuration Time

The configuration time corresponds to the time taken from the end of virtual machines provisioning until the Hadoop cluster is ready to start executing application. This is mainly determined by Resilin which performs the configuration of the Hadoop cluster.

We submit job flows on the service and compare the time for different cluster sizes from the end of provisioning until the start of the Hadoop computation. As it can be seen in Figure 3, the configuration time scales well with the number of instances to be configured.

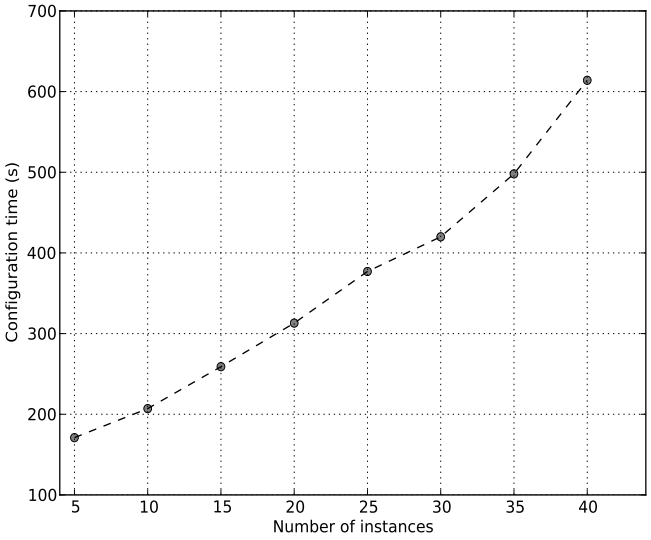



Figure 3: Configuration time for different cluster sizes using Resilin

## 4.4 Execution Time

For Resilin's evaluation, applications from different categories were executed. All experiments were done with intermediate and output data compression, thus minimizing network I/O utilization. All the virtual machines in the cluster were storing data for the applications. In the multiple clouds scenario, machines were distributed evenly on clouds. All the virtual machines involved in the experiments were also storing data for HDFS (core nodes). In the case of a distributed cluster, half of the number of virtual machines were in the OpenNebula cloud and the other half in the Nimbus cloud. The master node is ignored (placed in the Opennebula cloud for all the experiments).

Figure 4 presents the results of the execution time of Word Count. We ran a Word Count program on a collection of 110 GB of text files. These were generated using RandomWriter and RandomTextWriter programs contained in the Hadoop distribution.

The results obtained show similar execution time on one cloud and when using two clouds. As the shuffled and output data is much smaller that the input dataset, the disk and network utilization is low while the CPU utilization is high.

Figure 5 presents the results of the execution time of TeraSort. Terasort is compressing the shuffle and output data thus lowering the disk and network utilization [16]. The results of terasort execution also show a similar execution of the application.

Figure 6 presents the results for PageRank. PageRank consists of several jobs, with greater disk/network utilization than the application presented before. Thus, the results show a longer execution time on two clouds.

To show the elasticity of clusters built using Resilin, we run Wordcount with an input of 30 GB. Figure 7 presents the execution time in a static cluster and in an elastic cluster (scaled up during computation).

In conclusion, all the experiments done show a small performance degradation when using an execution platform built from resources provided by multiple clouds. This is due to lower

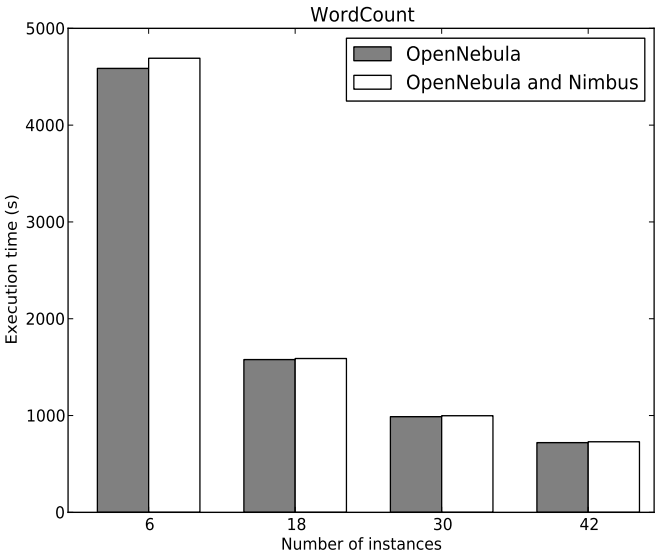



Figure 4: Execution time of Word Count on Resilin with different number of instances

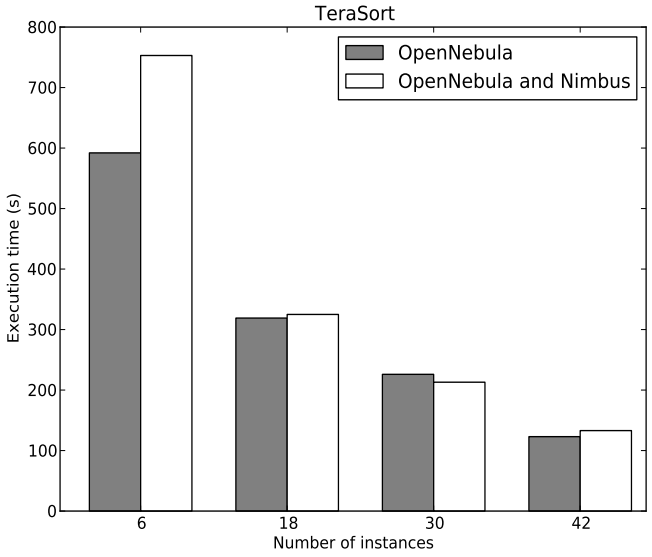



Figure 5: Execution time of TeraSort on Resilin with different number of instances

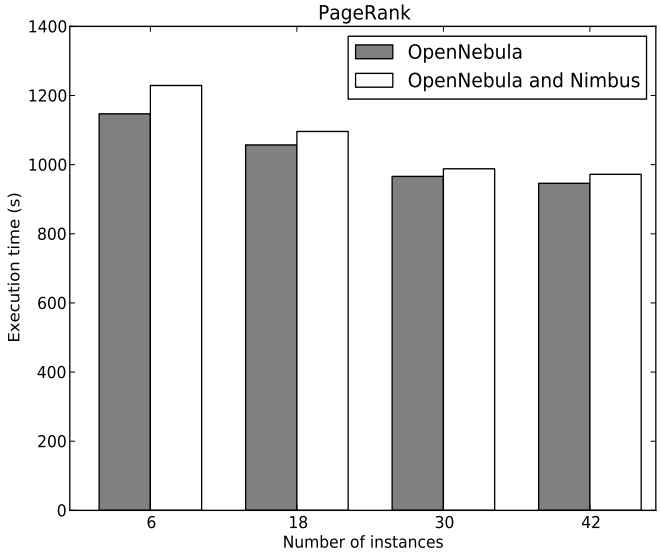



Figure 6: Execution time of Pagerank on Resilin with Hadoop clusters built from resources provided by one OpenNebula cloud and another one from OpenNebula and Nimbus clouds

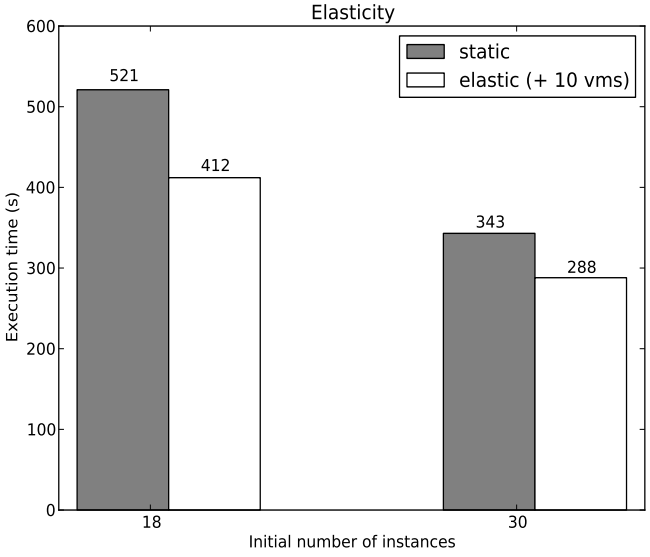



Figure 7: Performance when scaling up a Hadoop cluster.

network transfer rates between the virtual machines from different clouds.

# 5 Related Work

Related work on running MapReduce computations in the cloud has been focused on adapting MapReduce to cloud characteristics, either by creating completely new implementations of MapReduce, or by modifying existing systems. For instance, Gunarathne et al. [15] proposed AzureMapReduce, a MapReduce implementation built on top of Microsoft Azure cloud services [37]. They leverage several services offered by Azure, such as Azure Queues for scheduling tasks, Azure blob storage for storing data, etc. This allows their implementation to be decentralized and more elastic than a Hadoop-based cluster, leading to increased levels of fault-tolerance and flexibility. Similarly, Liu and Orban [20] created Cloud MapReduce, a MapReduce implementation leveraging services from the Amazon Web Services platform, with data stored in Amazon S3, and synchronization and coordination of workers performed with Amazon Simple Queue Service (Amazon SQS) and Amazon SimpleDB. These contributions are orthogonal to our objectives. While their goal is to build MapReduce implementations taking advantage of features offered by specific cloud services, we aim to bring an easy way to use MapReduce execution platform to many different cloud implementations. Furthermore, to our knowledge, no EC2-compatible open-source cloud implements the SQS and SimpleDB interfaces, making it impossible to run these MapReduce implementations outside of Microsoft Azure and Amazon Web Services.

Another type of proposition has been Seregenti, a toolkit developed by VMware [31]. It allows the fast deployment of highly available Hadoop clusters. Although it supports multiple Hadoop based distributions from a range of vendors including Apache Hadoop and Cloudera Distribution, Seregenti allows the usage of resources provided only by VMware vSphere platform.

An interesting and flexible approach is Apache Whirr, a library and command line tool that can be used to run cloud services. It simplifies the deployment of distributed systems on cloud infrastructure, allowing users to launch and tear-down complex cloud cluster environments with a single command. Similarly to Resilin, Whirr supports the configuration of Hadoop clusters but it does not expose a web service or support dynamic elasticity.

Another axis of research has been focusing on optimizing the number, type and configuration of resources allocated for a MapReduce computation. Kambatla et al. [17] study how to configure the number of map and reduce tasks running in parallel on each compute node. They propose to analyze the behavior of an application and to match it against a database of application signatures containing optimal Hadoop configuration settings, in order to derive efficient configuration parameters. To our knowledge, few works have studied the performance of large scale execution platforms built on top of multiple federation clouds. Moreno-Vozmediano et al. [23] executed Many-Task Computing applications on top of multiple clouds: a local infrastructure, two Amazon EC2 regions, and ElasticHosts [13]. However, they restricted their evaluation to small cluster sizes, and used simulation to extend their results to 256 worker nodes. Vockler et al. [36] executed an astronomy workflow application with an execution platform of 150 dual-core virtual machines provisioned from multiple FutureGrid clouds. They conclude that sky computing is a viable and effective solution for executing their application.

Elastic capabilities of cloud infrastructures are also a topic of interest in the community. The main area of focus has been on providing elasticity to web hosting platforms. These platforms leverage elasticity in order to adapt to changes in user traffic: peak hours, Slashdot effects, busy periods (Christmas holidays for online retail), etc. Amazon Web Services provides Auto Scaling [2], a service to scale up and down platforms built from Amazon EC2 resources. The Contrail

project develops ConPaaS [8], a Platform-as-a-Service layer providing a set of elastic high-level services [9] and runtime environments [10]. The high-level services include Scalarix, a scalable transactional key-value store, and SQL databases. Runtime environments allow MapReduce computations, web hosting of Java servlets and PHP applications, and execution of Bag of Tasks computations based on the BaTS algorithm [28]. These services and environments will provide elastic capabilities to respect their SLAs. Finally, several works have studied architectures and policies to extend local clusters with cloud resources [12] [21].

# 6 Conclusion and Future Works

In this paper, we presented Resilin, an implementation of the Elastic MapReduce API that allows users to run MapReduce computations on resources originating from other clouds than Amazon EC2, such as private and scientific clouds. Resilin takes care of provisioning Hadoop clusters and submitting jobs, allowing users to focus on writing their MapReduce applications rather than managing cloud resources. Resilin uses the EC2 API to interact with IaaS clouds, making it compatible with most open-source IaaS toolkits. We evaluated our implementation of Resilin on Nimbus and OpenNebula clouds deployed on the Grid'5000 testbed and compared the performance of jobflows executed on resources provided by one cloud provider and job flows executed on resources from multiple providers. As future work, we plan to complete compatibility with the Amazon Elastic MapReduce API, by providing HBase flows, increasing cloud providers range by implementing more connection interfaces(eg OCCI). We will also investigate how to improve the system by adding the feature to automatically scale the execution platform, based on a job flow profile.

For users having access to multiple clouds, we will investigate how Resilin could automatically select which cloud infrastructure to use, instead of relying on users to choose where they want to provision resources. Finally, we plan to release the Resilin software as open-source in the near future.

# References

[1] Amazon Elastic MapReduce. `http://aws.amazon.com/elasticmapreduce/`.

[2] Amazon Web Services. Auto Scaling. `http://aws.amazon.com/autoscaling/`.

[3] Apache Hadoop. `http://hadoop.apache.org/`.

[4] boto: A Python interface to Amazon Web Services. `http://boto.cloudhackers.com/`.

[5] John Bresnahan, Kate Keahey, David LaBissoniere, and Tim Freeman. Cumulus: An Open Source Storage Cloud for Science. In *2nd Workshop on Scientific Cloud Computing*, pages 25–32, 2011.

[6] Franck Cappello, Eddy Caron, Michel Dayde, Frédéric Desprez, Yvon Jégou, Pascale Primet, Emmanuel Jeannot, Stéphane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Benjamin Quétier, and Olivier Richard. Grid'5000: a large scale and highly reconfigurable Grid experimental testbed. In *6th IEEE/ACM International Workshop on Grid Computing*, pages 99–106, 2005.

[7] Michael Cardosa, Chenyu Wang, Anshuman Nangia, Abhishek Chandra, and Jon Weissman. Exploring MapReduce Efficiency with Highly-Distributed Data. In *MapReduce '11*, 2011.

[8] ConPaaS. `http://www.conpaas.eu/`.

[9] Contrail Project. D8.1: Requirements and Architecture for the implementation of the High Level Services. In *Deliverable 2011.*

[10] Contrail Project. Specification of requirements and architecture for runtime environments. In *Deliverable 2011.*

[11] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating Systems Design and Implementation*, pages 137–149, 2004.

[12] Marcos Dias de AssunÃ§Ã£o, Alexandre di Costanzo, and Rajkumar Buyya. Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters. In *18th International ACM Symposium on High Performance Distributed Computing*, pages 141–150, 2009.

[13] ElasticHosts. `http://www.elastichosts.com/`.

[14] FutureGrid. `https://portal.futuregrid.org/`.

[15] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. MapReduce in the Clouds for Science. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 565–572, 2010.

[16] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *22nd International Conference on,Data Engineering Workshops*, pages 41–51, 2010.

[17] Karthik Kambatla, Abhinav Pathak, and Himabindu Pucha. Towards optimizing hadoop provisioning in the cloud. In *USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–5, 2009.

[18] Katarzyna Keahey, Maurício Tsugawa, Andréa Matsunaga, and José A. B. Fortes. Sky Computing. *IEEE Internet Computing*, 13(5):43–51, 2009.

[19] Kate Keahey and Tim Freeman. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In *First Workshop on Cloud Computing and its Applications*, pages 1–5, 2008.

[20] Huan Liu and Dan Orban. Cloud MapReduce: a MapReduce Implementation on top of a Cloud Operating System. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 464–474, 2011.

[21] Paul Marshall, Kate Keahey, and Tim Freeman. Elastic Site: Using Clouds to Elastically Extend Site Resources. In *10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 43–52, 2010.

[22] Andréa Matsunaga, Maurício Tsugawa, and José Fortes. CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. In *4th IEEE International Conference on e-Science*, pages 222–229, 2008.

[23] Rafael Moreno-Vozmediano, RubÃ©n S. Montero, and Ignacio M. Llorente. Multi-cloud Deployment of Computing Clusters for Loosely Coupled MTC Applications. In *IEEE Transactions on Parallel and Distributed Systems*, pages 924–930, 2011.

[24] Nimbus. `http://www.nimbusproject.org/`.

[25] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, 2009.

[26] OpenNebula. `http://www.opennebula.org/`.

[27] OpenStack. `http://www.openstack.org/`.

[28] Ana-Maria Oprescu, Thilo Kielmann, and Haralambie Leahu. Budget estimation and control for bag-of-tasks scheduling in clouds. In *Parallel Processing Letters*, pages 219–243, 2011.

[29] paramiko. `http://www.lag.net/paramiko/`.

[30] Lavanya Ramakrishnan, Piotr T. Zbiegel, Scott Campbell, Rick Bradshaw, Richard Shane Canon, Susan Coghlan, Iwona Sakrejda, Narayan Desai, Tina Declerck, and Anping Liu. Magellan: Experiences from a Science Cloud. In *2nd Workshop on Scientific Cloud Computing*, pages 49–58, 2011.

[31] Seregenti. `http://serengeti.cloudfoundry.com/`.

[32] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

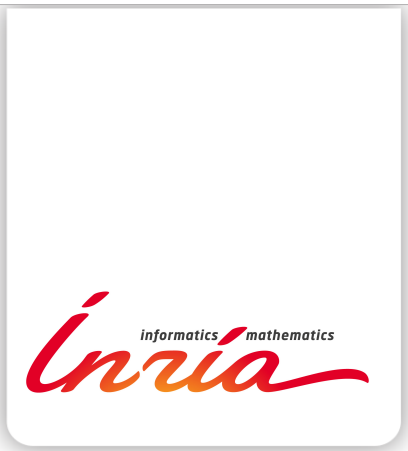
[33] Konstantin V. Shvachko. Apache Hadoop: The Scalability Update. *;login:*, 36(3):7–13, June 2011.

[34] Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sarma, Raghotham Murthy, and Hao Liu. Data Warehousing and Analytics Infrastructure at Facebook. In *2010 ACM SIGMOD International Conference on Management of Data*, pages 1013–1020, 2010.

[35] Twisted. `http://twistedmatrix.com/trac/`.

[36] Jens-S. Vockler, Gideon Juve, Ewa Deelman, Mats Rynge, and G. Bruce Berriman. Experiences Using Cloud Computing for A Scientific Workflow Application. In *ScienceCloud 2011*, pages 1–10, 2011.

[37] Windows Azure Platform. `http://www.microsoft.com/windowsazure/`.




**RESEARCH CENTRE**
**RENNES – BRETAGNE ATLANTIQUE**

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399